WHAT IS CLAIMED IS:

1. A method of on-the-fly patching of executable code comprising:

identifying original instructions to be changed;

copying the original instructions to a storage location;

adding a jump instruction to the copied instructions to return to a next instruction after the original instructions; and

replacing the original instructions with mark instructions and a transfer of control to a hook.

2. The method of claim 1, further comprising, prior to the allocating step, allowing a write operation on a page in memory where the original code is located.

3. The method of claim 1, further comprising, prior to the allocating step, masking interrupts.

4. The method of claim 1, further comprising, after the replacing step, disallowing a write operation on the page in memory where the block of code is located.

5. The method of claim 1, further comprising, after the replacing step, unmasking interrupts.

6. The method of claim 1, wherein the original instructions are changed in reverse order.

7. The method of claim 1, wherein the mark instructions are the same length, in bytes, as the instructions to be changed.

8. The method of claim 1, wherein the mark instructions are shorter in length, in bytes, as the instructions to be changed, and include NOP (no operation) filler.

9. The method of claim 1, wherein the modified instructions include a resolver to determine a number of the instructions at a location of the original code that had already been executed.

10. The method of claim 9, wherein the resolver determines a number of instructions that had already been executed using the mark instructions.

11. The method of claim 10, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be changed, the resolver calls the copied instructions at the storage location so as to imitate a "no patch installed" scenario.

12. The method of claim 11, wherein, after execution of the instructions at the storage location, the resolver returns control to the next instruction.

13. The method of claim 1, further comprising enabling functionality of the copied instructions at the storage location.

14. The method of claim 13, wherein the enabling step comprises reconciling addressing in the instructions in the storage location.

15. The method of claim 13, wherein the enabling step comprises alignment of instructions in the instructions at the storage location.

16. The method of claim 1, further comprising verifying that the original code is susceptible to patching.

17. The method of claim 16, wherein the verifying step determines whether any mark instructions are already present in the original instructions.

18. The method of claim 16, wherein the verifying step determines whether any copy protect instructions are already present in the original instructions.

19. The method of claim 16, wherein the verifying step determines whether the original instructions include a suitable jump point that can be modified to the transfer of control to the hook.

20. The method of claim 16, wherein the verifying step determines whether the original instructions represent valid instructions.

21. The method of claim 1, further comprising placing the hook in memory.

22. The method of claim 1, wherein the hook has been previously placed in memory.

23. The method of claim 1, wherein the replacing step uses an atomic write to replace the original instructions.

24. The method of claim 23, wherein the atomic write replaces one instruction at a time.

25. The method of claim 23, wherein the atomic write replaces multiple instructions at a time.

26. The method of claim 23, wherein, for Intel X32 architecture, the atomic write uses any of "xchg", "lock cmpxchg8b," "lock cmpxchg," and "lock xchg" instructions.

27. A method of on-the-fly patching of executable code comprising:

verifying that original instructions to be changed are susceptible to patching;

generating pseudooriginal code from the original instructions at a different storage location from the original instructions;

adding a jump instruction to the pseudooriginal code to return to a next instruction after the original instructions; and

replacing the original code with tag instructions that indicate only their execution and a transfer of control to a hook.

28. The method of claim 27, wherein the original instructions are changed in reverse order.

29. The method of claim 27, wherein the tag instructions are the same length, in bytes, as the instructions to be changed.

30. The method of claim 27, wherein the tag instructions are shorter in length, in bytes, as the instructions to be changed, and include NOP (no operation) filler.

31. The method of claim 27, wherein the modified instructions include a resolver to determine a number of the instructions at a location of the original code that had already been executed.

32. The method of claim 31, wherein the resolver determines a number of instructions that had already been executed using the tag instructions.

33. The method of claim 32, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be changed, the resolver calls the pseudooriginal code so as to imitate a "no patch installed" scenario.

34. The method of claim 33, wherein, after execution of the pseudooriginal code, the resolver returns control to the next instruction.

35. The method of claim 27, further comprising reconciling addressing in the instructions in the storage location.

36. The method of claim 27, further comprising verifying that the original code is susceptible to patching.

37. The method of claim 36, wherein the verifying step determines whether any tag instructions are already present in the original instructions.

38. The method of claim 27, further comprising placing the hook in memory.

39. The method of claim 27, wherein the replacing step uses an atomic write to replace the original instructions.

40. A method of on-the-fly patching of executable code comprising:

identifying original instructions to be changed;

allocating a storage location for storing a functionally equivalent copy of the original instructions;

copying the original instructions to the storage location; and

replacing the original instructions with mark instructions and a transfer of control to a hook.

41. The method of claim 40, further comprising, prior to the allocating step, allowing a write operation on a page in memory where the original instructions are located.

42. The method of claim 40, further comprising adding a jump instruction to the copied instructions to return control to a next instruction after the original instructions.

43. The method of claim 40, wherein the original instructions are changed in reverse order.

44. The method of claim 40, wherein the modified instructions include a resolver to determine a number of the instructions at a location of the original instructions that had already been executed.

45. The method of claim 44, wherein the resolver determines a number of instructions that had already been executed using the mark instructions.

46. The method of claim 45, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be changed, the resolver calls the functionally equivalent copy so as to imitate a "no patch installed" scenario.

47. The method of claim 46, wherein, after execution of the functionally equivalent copy, the resolver returns control to the next instruction.

48. The method of claim 40, further comprising verifying that the original instructions are susceptible to patching.

49. The method of claim 48, wherein the verifying step determines whether any mark instructions are already present in the original instructions.

50. The method of claim 48, wherein the verifying step determines whether any copy protect instructions are already present in the original instructions.

51. The method of claim 40, wherein the replacing step uses an atomic write to replace the original instructions.

52. The method of claim 40, further comprising enabling functionality of the copied instructions at the storage location.

53. A computer program product for on-the-fly patching of executable code, the computer program product comprising a computer useable medium having computer program logic recorded thereon for controlling at least one processor, the computer program logic comprising:

computer program code means for identifying original instructions to be changed;

computer program code means for copying the original instructions to a storage location;

computer program code means for adding a jump instruction to the copied instructions to return to a next instruction after the original instructions; and

computer program code means for replacing the original instructions with mark instructions and a transfer of control to a hook.

54. The computer program product of claim 52, wherein the original instructions are changed in reverse order.

55. The computer program product of claim 53, wherein the mark instructions are the same length, in bytes, as the instructions to be changed.

56. The computer program product of claim 53, wherein the mark instructions are shorter in length, in bytes, as the instructions to be changed, and include NOP (no operation) filler.

57. The computer program product of claim 53, wherein the hook includes a resolver to determine a number of the instructions at a location of the original code that had already been executed.

58. The computer program product of claim 57, wherein the resolver determines a number of instructions that had already been executed using the mark instructions.

59. The computer program product of claim 54, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be changed, the resolver calls the copied instructions at the storage location so as to imitate a "no patch installed" scenario.

60. The computer program product of claim 58, wherein, after execution of the instructions to the storage location, the resolver returns control to the next instruction.

61. The computer program product of claim 53, further comprising computer program code means for enabling functionality of the copied instructions at the storage location.

62. The computer program product of claim 61, wherein the computer program code means for enabling functionality of the copied instructions at the storage location comprises computer program code means for reconciling addressing in the instructions in the storage location.

63. The computer program product of claim 53, further comprising computer program code means for verifying that the original code is susceptible to patching.

64. The computer program product of claim 58, wherein the computer program code means for verifying determines whether any mark instructions are already present in the original instructions.

65. The computer program product of claim 53, further comprising computer program code means for placing the hook in memory.

66. The computer product of claim 53, wherein the computer program code means for replacing uses an atomic write to replace the original instructions.

67. A computer program product for on-the-fly patching of executable code, the computer program product comprising a computer useable medium having computer program logic recorded thereon for controlling at least one processor, the computer program logic comprising:

computer program code means for verifying that original instructions to be changed are susceptible to patching;

computer program code means for generating pseudooriginal code from the original instructions at a different storage location from the original instructions;

computer program code means for adding a jump instruction to the pseudooriginal code to return to a next instruction after the original instructions; and

computer program code means for replacing the original code with tag instructions that indicate only their execution and a transfer of control to a hook.

68. A computer program product for on-the-fly patching of executable code, the computer program product comprising a computer useable medium having computer program logic recorded thereon for controlling at least one processor, the computer program logic comprising:

computer program code means for identifying original instructions to be changed;

computer program code means for allocating a storage location for storing a functionally equivalent copy of the original instructions;

computer program code means for copying the original instructions to the storage location; and

computer program code means for replacing the original instructions with mark instructions and a transfer of control to a hook.

69. A system for on-the-fly patching of executable code comprising:

means for identifying original instructions to be changed;

means for copying the original instructions to a storage location;

means for adding a jump instruction to the copied instructions to return to a next instruction after the original instructions; and

means for replacing the original code with mark instructions and a transfer of control to a hook.

70. A system for on-the-fly patching of executable code comprising:

means for verifying that original instructions to be changed are susceptible to patching;

means for generating pseudooriginal code from the original instructions at a different storage location from the original instructions;

means for adding a jump instruction to the enabled pseudooriginal code to return to a next instruction after the original instructions; and

means for replacing the original code with tag instructions that indicate only their execution and a transfer of control to a hook.

71. A system for on-the-fly patching of executable code comprising:

means for identifying original instructions to be changed;

means for allocating a storage location for storing a functionally equivalent copy of the original instructions;

means for copying the original instructions to the storage location; and

means for replacing the original instructions with mark instructions and a transfer of control to a hook.